

Towards Run-Time Search for Real-World Multi-Agent Systems

Abigail C. Diller
Grand Valley State University
Allendale, Michigan, USA
dillerab@mail.gvsu.edu

Erik M. Fredericks
Grand Valley State University
Allendale, Michigan, USA
frederer@gvsu.edu

ABSTRACT

Multi-agent systems (MAS) may encounter uncertainties in the form of unexpected environmental conditions, sub-optimal system configurations, and unplanned interactions between autonomous agents. The number of combinations of such uncertainties may be innumerable, however run-time testing may reduce the issues impacting such a system. We posit that search heuristics can augment a run-time testing process, in-situ, for a MAS. To support our position we discuss our in-progress experimental testbed to realize this goal and highlight challenges we anticipate for this domain.

CCS CONCEPTS

• **Software and its engineering** → **Software testing and debugging; Software verification and validation**; • **Computing methodologies** → **Bio-inspired approaches**.

KEYWORDS

search-based software testing, multi-agent systems, cyber-physical systems

1 INTRODUCTION

Run-time testing can provide assurance that a multi-agent system (MAS) will continuously satisfy requirements and behave as expected even when faced with uncertainty [11]. However, MAS testing is a challenge as it must account for the unique characteristics of heterogeneous agents and may require different testing approaches [12, 13]. Furthermore, the number of potential combinations of system conditions, environmental interactions, and network issues that a MAS may experience during its life cycle may be impossible to enumerate [16]. We discuss the application of search-based software testing (SBST) techniques as an approach for navigating the complex solution space of MAS validation/verification tasks. We next present our work-in-progress experimental testbed, outline key challenges for the MAS domain, and highlight relevant research throughout the paper.

2 GREENROOM: EXPERIMENTAL TESTBED

A MAS is typically modeled as a network of agents (i.e., devices or simulated processes) that each act independently in support of their own goals while maintaining a common set of shared objectives [2, 8]. Agents monitor the operating context of both the system and environment and autonomously fulfill both local (i.e. impacting the agent itself) and global (i.e., impacting the system) objectives. Additionally, agents will generally communicate with other agents via message passing to ensure that knowledge is shared.

To support MAS execution, run-time requirements monitoring [15], run-time testing [5], and run-time verification [6, 10]

have been proposed as techniques for ensuring the system continuously satisfies key objectives (i.e., to ensure the MAS continuously expresses acceptable behaviors per its artifacts). Moreover, genetic algorithms [7] have been previously applied for hardware-based optimization in-situ for field-programmable gate arrays [14] as well as general search heuristics on constrained devices [4]. Similar to cyber-physical systems [3], a MAS can experience complex real-world concerns such as power consumption, human interaction, and unexpected weather conditions. Most similarly, Nguyen *et al.* explored evolutionary approaches for continuous MAS evolutionary testing [12, 13], however we focus on hardware applications in-situ and the related concerns therein (e.g., timing delays, lost/corrupted data, etc.).

Figure 1 shows GreenRoom, our hardware-based experimental testbed that both implements and extends our prior work [3]. We model our system as a MAS, where one agent (i.e., an Arduino Uno) is tasked with ensuring that the local environmental conditions are within an acceptable range (e.g., comfortable temperature, no dangerous gasses, etc.) and another agent (i.e., a Raspberry Pi 4) is tasked with continuously generating test data via a genetic algorithm [7] as well as monitoring power consumption of the system as a whole. Agents communicate via a serial connection to transmit/receive data as needed.

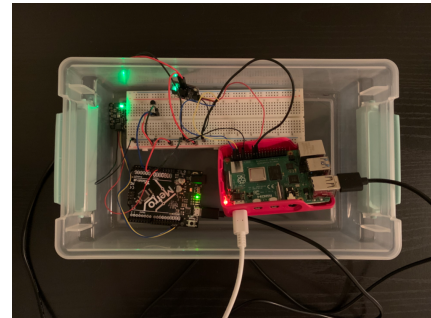


Figure 1: Experimental testbed comprising an Arduino Uno, Raspberry Pi 4, and wired sensors.

We next outline our procedure for performing SBST at run-time, comprising a *testing agent* (i.e., Raspberry Pi) and a set of *agents under test* (i.e., Arduino or other devices):

- (1) *Testing agent* configures search procedure at start of execution and continuously generates sets of test data relevant to the *current conditions* experienced by the MAS.
- (2) *Testing agent* packages test data and disseminates to target *agents under test*.
- (3) Each *agent under test* parses received test data, executes test case(s), and transmits results back to *testing agent*.

- (4) *Testing agent* collates test results, incorporates into fitness function (see Equation 1), and executes the subsequent generation of search.

While each search problem is different, we anticipate that a weighted fitness function can provide a lightweight method for guiding the search procedure. For GreenRoom, our proposed fitness function (i.e., Equation 1) comprises *maximizing* the number of test case failures and *maximizing* novelty of test data while intelligently exploring the solution space (i.e., via novelty search [9]). For the purpose of this paper, we abstract the sub-fitness functions for maximizing test failures and test data novelty, respectively.

$$ff = \alpha_{fail} * \max(TC_{fail}) + \alpha_{novelty} * \max(novelty) \quad (1)$$

Equation 2 presents a sample test case that checks the validity of monitored temperature sensor readings as well as test data generated by our testbed. At present, our Raspberry Pi is using EasyGA¹ to generate test data, a serial connection between the Raspberry Pi and Arduino to communicate, and injected sensor values to test the rigor of our bench setup.

$$TC_{temp}(t) = (t \geq -40 \ \&\& \ t \leq 85.0) ? \text{true} : \text{false} \quad (2)$$

Our genome comprises a list of values that reflect the input data of each test case. There are 20 test case templates currently implemented in GreenRoom leading to a small search space. We next highlight challenges we have faced thus far in applying SBST to our experimental testbed.

Challenge (1) - Search In-Situ: Testing a MAS during execution in its deployed environment has the benefit of using real-world data at the detriment of potentially impacting the system under test. As such, care must be taken to avoid disruption of critical system features/behaviors (e.g., system invariants).

Challenge (2) - Navigating the Search Space at Run Time: The search space must be efficiently explored with minimal resources to avoid impacting “normal” system operations. A lightweight search heuristic can enable online search (e.g., a (1+1) evolutionary algorithm [1]) to the detriment of searching power (e.g., local optima, premature convergence, etc.). As such, a fine balance between search capabilities and run-time impact must be considered.

Challenge (3) - Managing System Constraints: Similar to Challenge (2), agents within a MAS may be constrained devices that operate with minimal memory, storage, and power capabilities. For example, communication via serial connection between the Arduino and Raspberry Pi must follow strict timing constraints to avoid lost or garbled data. System capabilities and constraints must be considered when overlaying run-time testing techniques to avoid negatively impacting the agent(s) under test.

3 DISCUSSION

This paper has discussed early-stage research for GreenRoom, our experimental testbed for performing run-time SBST on a system that has been modeled as a MAS. At present, we have a preliminary hardware setup comprising multiple agents with individual goals, including generation of test data during execution and communication of test data/results between agents. We have also identified challenges that can be experienced by researchers in this domain. Future work for this project involves significant extension to our

body of test cases, incorporation of power monitoring metrics into our search procedure, and a full empirical evaluation.

ACKNOWLEDGMENTS

This work has been supported in part by grants from the Michigan Space Grant Consortium (#80NSSC20M0124) and Grand Valley State University. The views and conclusions contained herein are those of the authors and do not necessarily represent the opinions of the sponsors.

REFERENCES

- [1] N. Bredeche, E. Haasdijk, and A.E. Eiben. 2010. On-Line, On-Board Evolution of Robot Controllers. In *Artificial Evolution*, Pierre Collet, Nicolas Monmarché, Pierrick Legrand, Marc Schoenauer, and Evelyne Lutton (Eds.). Lecture Notes in Computer Science, Vol. 5975. Springer Berlin Heidelberg, 110–121.
- [2] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. 2004. Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems* 8, 3 (01 May 2004), 203–236.
- [3] Erik M. Fredericks, Kate M. Bowers, and Reihaneh H. Hariri. 2019. On incorporating search-based heuristics into real-world systems. In *Proceedings of the 12th Intl. Workshop on Search-Based Software Testing*. IEEE Press, 11–12.
- [4] Erik M. Fredericks and Jared M. Moore. 2020. Search@ Home: A Commercial Off-the-Shelf Environment for Investigating Optimization Problems. In *Intl. Symposium on Search Based Software Engineering*. Springer, 171–177.
- [5] Erik M. Fredericks, Andres J. Ramirez, and Betty H. C. Cheng. 2013. Towards run-time testing of dynamic adaptive systems. In *Proceedings of the 8th Intl. Symposium on Software Engineering for Adaptive and Self-Managing Systems* (San Francisco, CA, USA) (SEAMS ’13). IEEE Press, 169–174.
- [6] C. Ghezzi. 2010. Adaptive Software Needs Continuous Verification. In *2010 8th IEEE Intl. Conference on Software Engineering and Formal Methods (SEFM)*. 3–4.
- [7] John H. Holland. 1992. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA.
- [8] Manuel Kolp, Paolo Giorgini, and John Mylopoulos. 2002. A Goal-Based Organizational Perspective on Multi-agent Architectures. In *Intelligent Agents VIII*. 128–140.
- [9] Joel Lehman and Kenneth O Stanley. 2011. Novelty search and the problem with objectives. In *Genetic programming theory and practice IX*. Springer, 37–56.
- [10] Yoo Jin Lim, Gwangui Hong, Donghwan Shin, Eunyoung Jee, and Doo-Hwan Bae. 2016. A runtime verification framework for dynamically adaptive multi-agent systems. In *2016 Intl. Conference on Big Data and Smart Computing (Big-Comp)*. IEEE, 509–512.
- [11] Henry Muccini, Mohammad Sharaf, and Danny Weyns. 2016. Self-adaptation for cyber-physical systems: a systematic literature review. In *Proceedings of the 11th international symposium on software engineering for adaptive and self-managing systems*. 75–81.
- [12] Cu D. Nguyen, Simon Miles, Anna Perini, Paolo Tonella, Mark Harman, and Michael Luck. 2012. Evolutionary testing of autonomous software agents. *Autonomous Agents and Multi-Agent Systems* 25, 2 (2012), 260–283.
- [13] Cu D. Nguyen, Anna Perini, Paolo Tonella, and Fondazione Bruno Kessler. 2007. Automated Continuous Testing of MultiAgent Systems. In *The Fifth European Workshop on Multi-Agent Systems (EUMAS)*.
- [14] Murat Peker. 2018. A fully customizable hardware implementation for general purpose genetic algorithms. *Applied Soft Computing* 62 (2018), 1066–1076.
- [15] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein. 2010. Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems. In *Requirements Engineering Conference (RE), 2010 18th IEEE Intl. 95–103*.
- [16] Bang Xiang Yong and Alexandra Brintrup. 2019. Multi agent system for machine learning under uncertainty in cyber physical manufacturing system. In *Intl. Workshop on Service Orientation in Holonic and Multi-Agent Manufacturing*. Springer, 244–257.

¹See <https://pypi.org/project/EasyGA/>.